BLUE CHiP

# The Configurable, Highly Parallel (CHiP) Approach for Signal Processing Applications

by

Lawrence Snyder

The BLUE CHiP Project
Purdue University
Department of Computer Sciences
Math Sciences Building
West Lafayette, Indiana 47907

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

| REPORT DOCUMENTATION PAGE | | READ INSTRUCTIONS BEFORE COMPLETING FORM |
|---|---|---|
| 1. REPORT NUMBER<br>CSD-TR-402 | 2. GOVT ACCESSION NO.<br>AD-A119 116 | 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE (and Subtitle)<br>The Configurable, Highly Parallel (CHiP) Approach for Signal Processing Applications | | 5. TYPE OF REPORT & PERIOD COVERED<br>Technical, Interim |
| | | 6. PERFORMING ORG. REPORT NUMBER |
| 7. AUTHOR(s)<br>Lawrence Snyder | | 8. CONTRACT OR GRANT NUMBER(s)<br>N00014-80-K-0816<br>N00014-81-K-0360 |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS<br>Purdue University<br>Department of Computer Sciences<br>West Lafayette, Indiana 47907 | | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS<br>SRO-100 |
| 11. CONTROLLING OFFICE NAME AND ADDRESS<br>Office of Naval Research<br>Information Systems Program<br>Arlington, Virginia 22217 | | 12. REPORT DATE<br>May 1982 |
| | | 13. NUMBER OF PAGES<br>17 |
| 14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) | | 15. SECURITY CLASS. (of this report)<br>Unclassified |
| | | 15a. DECLASSIFICATION DOWNGRADING SCHEDULE |

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this report is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

design methodology, CHiP computer, switch lattice, FFT, shallow hierarchy, graph embeddings VLSI design methodology, signal processing

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A VLSI design methodology, built around the CHiP architecture, is described. The switch lattice of the CHiP architecture is the primary design abstraction. The lattice is a flexible design medium with constraints that mirror those of raw silicon. An eight point pipelined Fast Fourier Transform design, used as a running example, is of independent interest for its locally connected layout.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

# The Configurable, Highly Parallel (CHiP) Approach for Signal Processing Applications

*Lawrence Snyder*

Department of Computer Sciences
Purdue University
West Lafayette, Indiana 47907

**CSD-TR-402**

*ABSTRACT*

A VLSI design methodology, built around the CHiP architecture, is described. The switch lattice of the CHiP architecture is the primary design abstraction. The lattice is a flexible design medium with constraints that mirror those of raw silicon. An eight point pipelined Fast Fourier Transform design, used as a running example, is of independent interest for its locally connected layout.

The work described herein is part of the Blue CHiP Project and is supported in part by the Office of Naval Research Contracts N00014-80-K-0816 and N00014-81-K-0360. The latter is Special Research Opportunities Task SRO-100.

This paper appeared in the Proceedings of the Technical Symposium East, Society of Photo-Optical Instrumentation Engineers, 1982.

# The Configurable, Highly Parallel (CHiP) Approach for Signal Processing Applications

*Lawrence Snyder*

Department of Computer Sciences
Purdue University
West Lafayette, Indiana 47907

**CSD-TR-402**

## Introduction

Between the conception of a real time signal processor and its functional VLSI realization there is an enormous amount of effort devoted to designing, revising, optimizing and testing. Since the process is cumulative -- later work builds on previous work -- and since the activity becomes progressively more detailed, more constrained and more exacting, it follows that *the global design parameters should be fully explored*. Global design decisions, when correct, can have a greater effect on performance than many local optimizations. When the decisions are wrong, they can cause continual difficulty. Accordingly, we propose a design methodology based on the Configurable, Highly Parallel (CHiP) architecture family [1] that focuses on exploring global design parameters and is especially well suited to the VLSI implementation of signal processing systems.

The characteristic that distinguishes digital signal processing design problems from other large VLSI design problems, e.g., microprocessor design, is that the former tend to require the assembly of a large number of identical components while the latter often require the assembly of a diverse collection of components. In terms of the widely discussed hierarchical design methodology [2-4], this distinction means that signal processors are characterized by a shallow hierarchy rather than a deep hierarchy. The emphasis on decomposition in the hierarchical design methodology with its resulting deep hierarchy provides less leverage for signal processing design problems. Our CHiP computer methodology, though hierarchical, emphasizes the layout of homogeneous components and should provide greater leverage for signal processor design situations.

The methodology is not a cookbook procedure. That is, there is not a sequence of definite steps which if followed from start to finish result in a real time signal processor. But there are steps: the designer programs the algorithm for a CHiP computer, tests it, assesses the design, revises it, programs the subparts, tests them, assesses their design, revises them and, finally, specializes the entire system for silicon implementation.

In order to organize our presentation of the methodology, we will develop a design as a running example. Our problem will be to design a pipelined, eight point Fast Fourier Transform processor. The reader need not be acquainted with the FFT, since our intent is not to produce a practical device. Rather we are using the problem as a context in which to focus on the design activity.

## Problem Statement

Naturally, the first step in any design situation is to understand the problem. For our running FFT example this can be conveniently stated with a schematic diagram, (Figure 1). Each processing element takes two inputs, $B$ and $B'$ and computes two weighted sums, $B + QB'$ and $B - QB'$. (See Stone [5] for exact details.) Our assumptions are that the processors receive data bit-serially from off the chip, that the structure is pipelined and that the resulting circuit is to be placed on a single chip. From these assumptions, we conclude that we will need to place twelve processors each capable of multiplying by a constant and adding, and that the chip will require sixteen pins for data in addition to power, ground and any control lines.

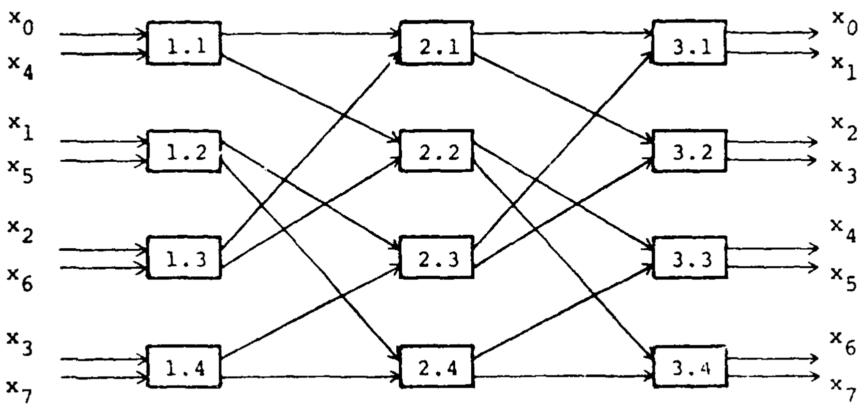


Figure 1. Pipelined FFT schematic.

## Programming the Algorithm

The next step in the methodology is to program the algorithm for a CHiP computer. The purpose is to establish an unambiguous specification of the problem and to begin initial exploration of the layout, timing and input/output constraints. Before programming our FFT example, we must introduce CHiP machines.

A CHiP computer is one of a family of architectures specialized for "fine-grained" parallelism and efficient VLSI implementation. The main component of the architecture (and the only one of interest here*) is the *switch lattice*. This is a homogeneous array of programmable switches

*Other, more thorough descriptions of the CHiP machine have been given, but they focus on its use as a general purpose parallel processor [1, 6]. Our description here has been specialized to its use as a design abstraction.

and data paths with processing elements (PEs) placed at regular intervals. Figure 2 illustrates schematic diagrams of two switch lattices. The switches and data paths are a general means of specifying information flow and the processing elements serve to represent some arbitrary computational activity.

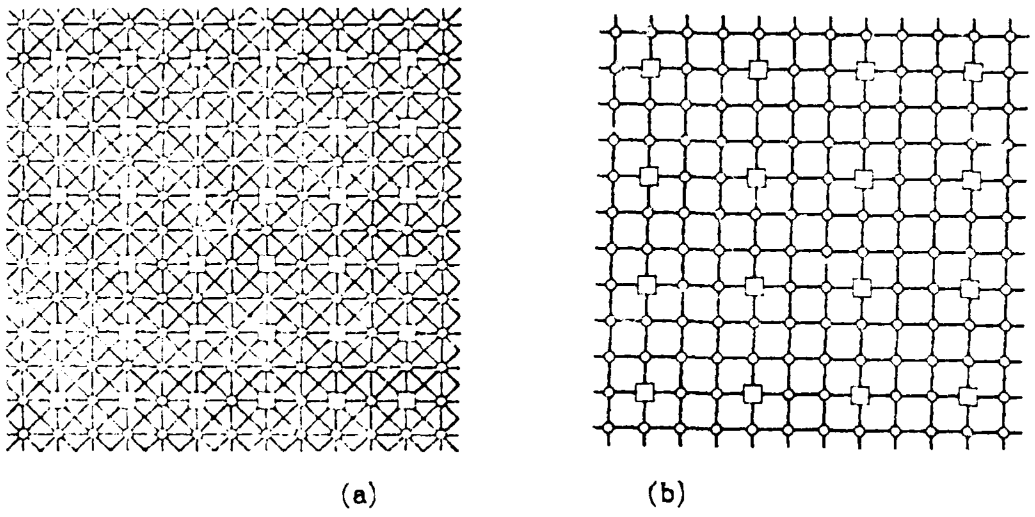


Figure 2. Two switch lattices. Circles are switches; squares are processing elements.

Ultimately, when the methodology has been worked through and the design is completed, the switches and data paths will have been removed, the active data paths will have been replaced by wires and the processing elements will have been replaced by specialized circuits for the particular function. But at this point this stylized representation of the components gives the designer a simple, flexible means of simulating the algorithm. The simplicity and flexibility make the revision a less painful process and encourage exploration and experimentation.

As Figure 2 illustrates, switch lattices differ in several respects. Although the designer will choose a lattice suitable for the particular algorithm, it is appropriate to mention the axes of variability. The degree, $d$, of switches and processing elements refers to the number of data paths incident to the device. Normally, we will have $d=8$ for switches and PEs although a higher degree for PEs may be convenient when there are multiple inputs and outputs. (See below.) In Figure 2(a), $d=8$ and in Figure 2(b), $d=4$.

The corridor width $w$ refers to the number of switches separating two neighboring PEs. (In Figure 2(a), $w=1$; in 2(b), $w=2$.) The more distinct data paths that must pass between two processing elements, the wider the corridor width must be. Since the switches will ultimately be removed, there is no harm in specifying a large corridor width. However, by calling explicit attention to corridor width, we cause the designer to focus on data routing and to appreciate the consequences of haphazard routing on density and packing. Notice that the corridor width is related to the number of distinct data paths passing between two PEs, not to the number of wires in each data path (which is set later).

One programs the switch lattice simulator by giving "configuration settings" for the switches and program text for the PEs. A configuration setting specifies which of the incident data paths a switch is to connect. If no configuration setting is given the data paths are isolated. In the figures we simply draw lines through switches to specify active settings. The program text is given in a conventional sequential programming language that has been extended with facilities to specify timing.*

Returning to our FFT example, we can specify our first embedding. Figure 3 illustrates a direct embedding of the FFT interconnection (Figure 1) in a switch lattice where $w=2$ and $d=8$. Because of the number of data paths crossing from the upper half of the layout to the lower half, a

*For the Blue CHiP Project's pilot simulator, the language is Pascal.

width $w=2$ is required. Notice that the layout is the same for each of the three files.

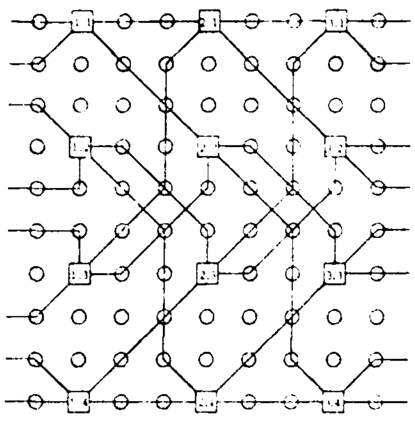


Figure 3. Switch lattice embedding of the FFT.

The execution of the CHiP computer is synchronous, so the development of the PE code is a simple matter. Each PE executes a variant of

```
L:  READ B; READ B'
    C ← B + QB'
    C' ← B − QB'
    WRITE C; WRITE C'
    GOTO L
```

where the variant is determined by which PE ports the variable comes from or goes to. For example, PE 1.1 would execute

```
L:  READ B FROM West; READ B' FROM Southwest
    C ← B + QB'
    C' ← B − QB'
    WRITE C TO East; WRITE C' TO Southeast
    GOTO L
```

PEs with degree greater than eight have their ports numbered.

Although the development of the program is the responsibility of the designer, there are library embeddings available that embody careful analysis and research.

## Assessment and Revision

The next activity in the methodology is to assess the initial design and make appropriate improvements. The goal here is to evaluate how the design can be globally improved before investing any effort in the detailed layout. Obviously, this activity will require a certain amount of judgement and experience.

Our FFT has several favorable characteristics. It has a nearly square aspect ratio (4:3) and has edge-to-edge data flow. The latter property is important in order to reach the bonding pads which are most conveniently located on the perimeter. The main liability of our initial design is the nonlocal data flow, i.e., the presence of long data paths. When the design is laid out, some wires will have to be as long as the side of a PE.

To solve this long data path problem, we observe that to achieve edge-to-edge data flow, it is *not* necessary for the flow to be unidirectional as it is in our initial design. In particular, an alternative strategy is to route the data towards the center of the layout and then back out towards the perimeter. To achieve such an in-and-out data flow, we place the second file ($2x$) of processing elements in the center of the layout and place the first and third files around the edge. Figure 4 illustrates this layout. The result is a design which still has edge-to-edge data flow and short, local connections. (This particular optimization may not generalize for larger shuffle graph problems, but the concept of in-and-out, edge-to-edge data flow could have wide application.)

The assessment and revision activity is iterated.

In the second design the aspect ratio is now square -- a minor improvement. Unfortunately, the corners of the layout are unused. This area can be used for bonding pads for the input/output wires of the adjacent PEs. It could also be used for other logic depending on how the design develops. (See below.)

When studying the way data enters and leaves the PEs in Figure 4, one sees that there are two different processing element geometries: The external PEs are alike and the internal PEs are alike*. It is obviously undesirable to have to require two designs for the same function, so we reprogram the external switches to convert the external PE geometries to the internal form. (See Figure 5.) This gives one layout form. Furthermore, if we reflect on how this stylized diagram will finally be implemented, it is clear that since the two data paths will probably *exit* from a PE together, the global data flow will be optimized if (1) they enter the PE together and (2) these entry and exit points are on opposite corners of a PE. We take these two conditions as constraints to be carried over to the next phase of design. If we can accomplish these two in the next phase we will have a better global organization. If we cannot, we return to this point to reassess and revise.

## Round Two

The process of programming the CHiP machine has resulted in an unambiguous specification of the algorithm, a routing of the data flow, a global layout and, presumably, the development of some test data that was used when the algorithm was run on the CHiP architecture simulator. But this first program is not intended to specify the algorithm in great enough detail for direct VLSI design and layout. In particular, the functional activity of the PEs is probably too complicated at this early stage.

*The internal PEs are not quite alike -- the (clockwise) meaning of the data paths differs among them. This will be easily corrected later by a simple wire crossover.

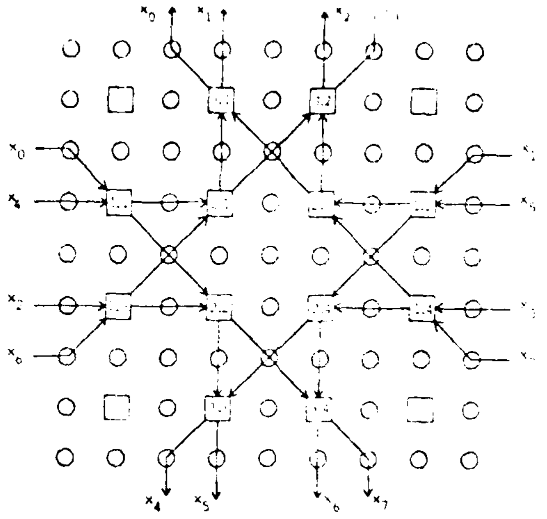


Figure 4. A revised FFT embedding with local communication.

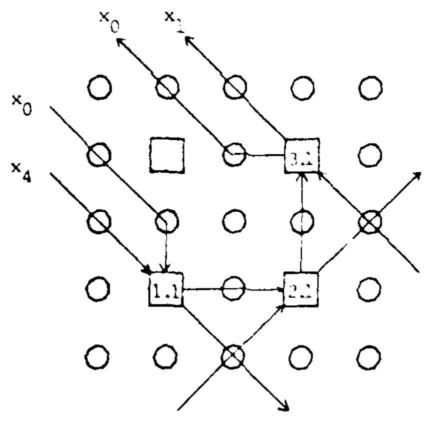


Figure 5. A reprogramming of perimeter interconnections.

In our FFT example, the inner product step is such a complicated activity.

So the methodology dictates that we iterate the program-assess-revise cycle until the functional activity of the PEs is sufficiently simple to be directly implemented in VLSI or can be implemented by an available library layout. Since the interconnection and global layout are now fixed, it is necessary only to implement the specified activity of the PE. This is accomplished by programming the CHiP architecture to implement the algorithm specified by the PE code(s). It is this iterative activity that gives the methodology its hierarchical capability.

During each subsequent round of programming-assessment-revision, it is important to establish that the current CHiP program correctly implements the specification of the previous level. This is a requirement of any top-down design effort, and it is aided here by the previously developed test data. (Notice that the test data may have to have its *form* changed to reflect the changed level of detail. For example, at one level the program can be simulated on words of data while at the next level it might require bit-serial data.)

We return briefly to the FFT example to give a second level of layout. Postulate a linear array of PEs to perform the inner product step based on a pipelined multiplier [7]. The layout will have two serial inputs, $B$ and $B'$, and will produce two serial outputs, $B + QB'$ and $B - QB'$. The coefficient, $Q$, will be stored internally to the layout, although it will be shifted through to form the intermediate products. By our analysis from the previous level, the current layout will receive its input at one corner and must deliver its output to the opposite corner. This suggests a "snaked" arrangement for the linear array of processing elements. (See Figure 6.) Each PE has a input and output the three data values as well as the partial product. The $B$ value is carried along to be available at the end for summing and differencing in the last cell. The control lines could either be broadcast or transmitted sequentially [7] from the control circuit that we will place in the corners of the global design.

As before we should next program the activity of the PEs. This time

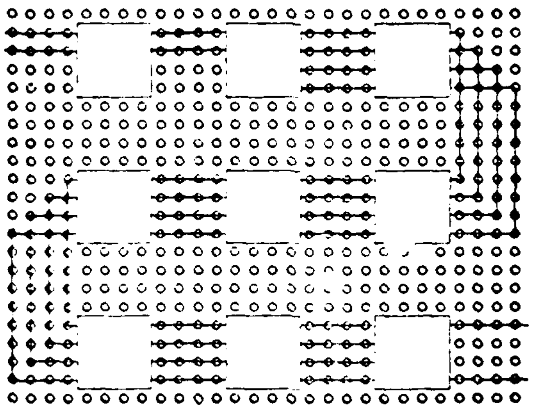


Figure 6. A pipelined inner product layout

there will be a few different cells since the multiplier requires a few [7] and there is a sum/difference cell. Then we embark on another sequence of assess and revise iterations. Having illustrated how the "opposite corner" data flow property established at the top level becomes a constraint to be implemented at the second level, we forego further detailed design.

## Design Specialization

The program-assess-revise cycle continues until the processing performed at each PE can be directly implemented as a VLSI design. The needed cells are either produced or acquired from a library. Then the design is *specialized*. That is, the VLSI designs replace the PEs in the last CHiP program layout. The active data paths are replaced by wires and all of the switches are removed. This result is then used to specialize the next higher level program, i.e., it replaces the PEs in its predecessor layout, etc. When the activity is completed, our stylized CHiP lattice is gone and what remains is a completed VLSI design. For our example, see the schematic in Figure 7.

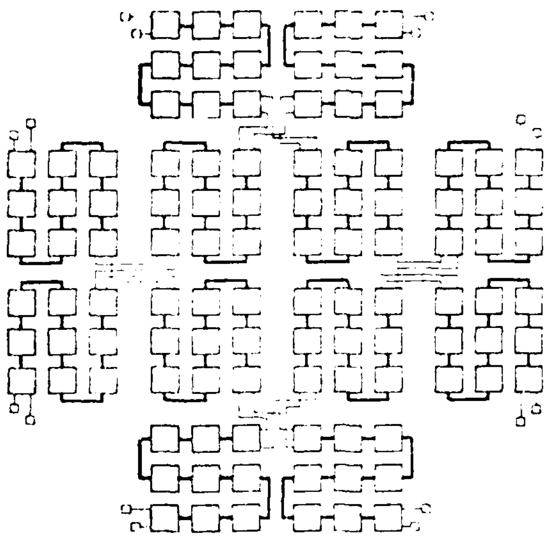


Figure 7. The specialized layout; the wide spacing is for showing interconnections.

Although it is straightforward, the specialization process is not quite as *trivial as just suggested. Its success depends on several conditions.* First, the aspect ratios and cell sizes must be properly controlled during the design process in order to pack the cells easily. This condition is easily met as long as the PEs perform closely related operations. In our running example, the top level cells were identical; the second level cells were sufficiently similar to justify an assumption of equal size.

Another complication for specialization is power and ground routing. We recommend the following strategy. Perform the routing prior to specialization but after all the VLSI cells are designed. At that point it is known, relatively, where power and ground enter the cells. Then, route the power and ground wires within each CHiP lattice layout starting at the *top* level. This permits a convenient top-down routing with the added advantage of knowing the target sites for the bottom level connections.

A word about simulation. As the program-assess-revise cycle is performed, each program can be simulated in isolation using the data (possibly revised) from the previous level. Moreover, the composit design can be simulated at each cycle by logically substituting the programs of each level for the PEs of the previous level. Once the PEs have been replaced by VLSI cells, however, it is unclear to what extent the design methodology can assist in efficient simulation. It is obviously compatible with hierarchically-based VLSI design rule checking [8] and electrical integrity checking [9].

## Summary and Discussion

The methodology we have presented focuses on global design issues of a VLSI implementation - data flow, functional decomposition, geometric layout of components. If we use '+' to denote 'one or more applications of', then the CHiP architecture methodology could be described as

$$(program\,,\ test\ (assess\,,\ revise\,,\ test\,)^{+})^{+}\ specialize$$

This methodology leads to a design with a shallow hierarchy, making it most effective for highly regular algorithms such as digital signal processing systems.

The CHiP architecture is crucial to the methodology. The switch lattice provides a medium that mirrors raw silicon: it is planar; it has integrated processing and interconnection facilities; it is described geometrically; external data is available only at the perimeter. Consequently, programming an algorithm for a CHiP architecture, though reasonably convenient, gives a good approximation to a VLSI layout.

It is this feature, a convenient programming abstraction imposing VLSI-like constraints, that perhaps most distinguishes the CHiP methodology from others in which the specification form is divorced from the technology.

## Related Results

There are three points to be made about related research.

First, from our study of configuration settings we have developed a library of efficient embeddings for commonly used interconnection structures. These include single corridor, planar, linear area binary trees [1, 10], toruses with no long data paths [10], shuffle-exchange graphs with narrow corridors, etc. For example, Figure 8 shows a 64 node shuffle-exchange graph embedded in a lattice with $w=1$ and $d=8$. This embedding, due to Paul Morrissett [11] is of interest because, in general, the shuffle-exchange graph requires very wide corridors [6]. In addition, there are general embedding techniques known for common layout problems: the Aleliunas-Rosenberg technique for bending data paths around corners [1], and lacing for maximizing the number of data paths through a region of the graph [10].

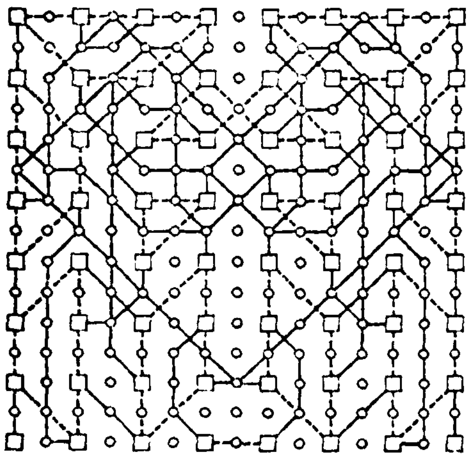


Figure 8. A 64 node shuffle-exchange graph.

Second, we have developed another methodology, called Processor Displacement, that assists the designer in balancing pin limitations with chip area utilization [12]. This approach to determining the optimal amount of multiplexing is compatible with the CHiP architecture methodology described here.

Third, the CHiP computer is intended to be a general purpose parallel processor and as such it physically implements a switch lattice with programmable switches and microprocessors as processing elements [1]. Were CHiP computers generally available, a signal processing system could be built simply by running the top level program of our methodology. This solution to constructing a special purpose signal processor probably would not have sufficiently good performance to serve most applications. Although easily accomplished, this would be too general a solution for a high performance device. Our methodology on the other hand can lead to high performance but requires much effort. There could be a compromise solution: We are exploring the possibility of *semispecialized* CHiP computer which would replace the general purpose microprocessor PEs with functional units tailored to a specific application. CORDIC processors are good candidates for these specialized PEs [13].

**Acknowledgments**

It is a pleasure to thank Stephan Bechtolsheim and Dennis Gannon for helpful discussions on these topics. Thanks are due to Julie Hanover for the expert manuscript preparation.

## References

[1] Lawrence Snyder
Overview to the Configurable, Highly Parallel Computer
*Computer*, 15 (1): 47-56, January 1982.

[2] J. A. Rowson
Understanding Hierarchical Design
Ph.D. Thesis, California Institute of Technology, 1980

[3] J. Craig Mudge
VLSI Chip Design at the Crossroads
In John P. Gray, editor, *VLSI 81*, Academic Press, pages 205-215, 1981

[4] R. H. Krambeck, D. E. Blahut, H. F. S. Law, B. W. Colbry, H. C. So, M. Harrison and J. Soukup
Top Down Design of a One Chip 32-bit CPU
In John P. Gray, editor, *VLSI 81*, Academic Press, pages 35-41, 1981

[5] Harold S. Stone
Parallel Processing with the Perfect Shuffle
*IEEE Transactions on Computers*, C-20(2): 153-161 (1971)

[6] Lawrence Snyder
Overview of the CHiP Computer
In John P. Gray, editor, *VLSI 81*, Academic Press, pages 237-246, 1981

[7] R. F. Lyon
Two's Complement Pipeline Multipliers
*IEEE Transactions on Communication*, COM-24, pages 418-425, April 1976

[8] Telle Whitney
A Hierarchical Design Analysis Front End
In John P. Gray, editor, *VLSI 81*, Academic Press, pages 217-225, 1981

[9] S. C. Johnson
Hierarchical Design Validation Based on Rectangles
In Paul Penfield, Jr., editor, *Proceedings of the MIT Conference on Advanced Research in VLSI*, Artech House, pages 97-100, 1982

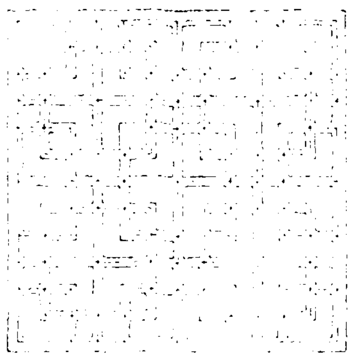
[10] Lawrence Snyder
Programming Processor Interconnection Structures
Technical Report CSD-381, Purdue University, 1981

[11] Paul Morrissett
Observations on Graph Embeddings
Blue CHiP Project Notes, November, 1981

[12] David M. DeRuyck, Lawrence Snyder and John D. Unruh
Processor Displacement: An Area-Time Trade-Off Method for VLSI
In Paul Penfield, Jr., editor, *Proceedings of the MIT Conference on*

*Advanced Research in VLSI*, Artech House Books, pages 182-187, 1981

[13] Hassan M. Ahmed, Jean-Marc Delosme and Martin Morf
Highly Concurrent Computing Structures for Matrix Arithmetic and Signal Processing
*Computer*, 15(1): 65-82, January 1982



Type lattice for the "In-Out Bond" lattice class; origin unknown.

ATE

LME